GROWBOT

Manual Rev. 1.1 for Rev. C Circuit Board (Rev. A is shown on cover photo)

The Parallax GrowBot is a mobile robot kit based on the BS2-IC module. The kit includes the parts required to install two LEDs, left and right front bumpers, two phototransistors, and a pushbutton. In addition, there is a small speaker and servo drivetrain on board. The standard header accommodates Parallax AppModules[1] for additional access to the BS2-IC's I/O pins. This documentation shows you how to assemble, program, and customize your GrowBot. This documentation will be frequently updated in Adobe PDF format on the Parallax web site http://www.parallaxinc.com.

Table of Contents

Chapter 1: Packing List 2
Chapter 2: Tools Required 4
Chapter 3: Soldering Tips 4
Chapter 4: GrowBot PCB Assembly 5
Chapter 5: GrowBot PCB Testing 12
Chapter 6: Servo Modification 13
Chapter 7: Final Assembly 15
Chapter 8: Program the GrowBot 16

Appendix A: GrowBot Schematic 18
Appendix B: GrowBot Programming Tips and PBASIC Quick Reference Guide 20

Try the following Parallax AppMods:

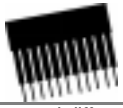
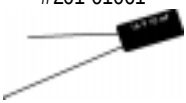
Through-hole Prototype Board (#29110)
Breadboard (#29114)

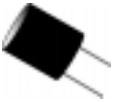
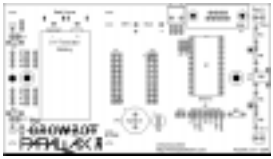
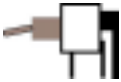
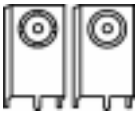
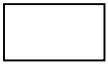
[1] AppModules are compatible with the Parallax BASIC Stamp Activity Board (#27905)

GrowBot starts here!

Chapter 1: Packing List

Before building your GrowBot, verify that you have all the parts. Parallax stock codes are provided in case you need to order a spare or replacement part. The term "non-stock" means you'll need to describe the piece when you order it since we don't assign a part number in our inventory system.

Three (3) 10K Ohm 1/8W 5% brow-black-orange #150-01031	Four (4) 220 Ohm 1/8W 5% red-red-brown #150-02211	One (1) 24 pin socket #450-02401
One (1) 220K Ohm 1/8W 5% red-red-yellow #150-03911	One (1) 100K Ohm 1/8W 5% brown-black-yellow #150-01041	One (1) tact button switch #400-00002
One (1) red snap-on button #400-00005	Two (2) tall headers #450-02002 	Two (2) 2-pin SIP headers #451-00201
One (1) 7-pin SIP header #451-00701	Two (2) LED red diffused T 1-3/4 #350-00006	Two (2) flathead screws 3/8" phillips 4-40 #700-00016
One (1) 10uF 16V aluminum electrolytic capacitor #201-01061 	Five (5) 0.1 uF radial Capacitors #200-01040	One (1) DB-9 connector #452-00005

One (1) sound generator #900-00001 	Two (2) Futaba S-148 Servos #900-00003 	One (1) antennae wire 0.026" 303 stainless #700-00010
Two (2) drive wheels #700-00013 	Two (2) stand offs, 4-40 ¾" alum. #700-00008 	BASIC Stamp II #BS2-IC
One (1) 4 AA battery holder 700-00005 	One (1) GrowBot PCB #300-15001 	One (1) GrowBot disk #950-00003
One (1) serial cable #800-00003 	One (1) switch #400-00013 	Two (2) o-ring wheels #700-00011
Two (2) photoresistors #350-00009 	One (1) set of 9-volt battery clips #452-00013 and -14 	One (1) ball tail wheel assembly #700-0009 and -14
Foam tape square ¾" non-stock 	One (1) #4 nylon washer non-stock 	Sticky back velcro 4" X00-00010

Chapter 2: Tools Required

Building the GrowBot requires the following tools not included in this kit (Radio Shack part numbers are included in case you want to buy these pieces locally):

1. Soldering iron (RS #64-2051)
2. Solder (RS #64-005)
3. Small Phillips screwdriver
4. Small cutters (RS #64-1833)
5. Small needle-nose pliers

Once the GrowBot is built and you are ready to program and run, you'll need an IBM PC compatible computer running DOS or Windows, four (4) AA alkaline or NiCad batteries, and one (1) 9-volt battery.

Chapter 3: Soldering Tips

Attaching components to the GrowBot's PC board requires soldering. If you haven't soldered before, there are some great tips included in Scott Edwards' *Programming and Customizing the BASIC Stamp Computer*[2]. Most importantly, don't eat while soldering and always wear safety glasses. Below is a consolidated summary of Scott's other soldering tips.

1. Moisten the sponge with water so it is damp and not soaking wet. Put the iron in the stand and plug it in. Don't touch the heating element. If you do painful burns and the stink of burned skin will remind you not to do it again.
2. When the iron is hot, take it out of the stand and wipe the tip on the sponge with a striking-a-match quick motion.
3. Apply a little solder to the tip of the soldering iron. If it beads up, wipe the iron on the sponge again and reapply. This is called "tinning".
4. Bring the freshly tinned tip of the iron into contact with the wires / metals you want to solder. The tinning of the tip will help transfer heat into the target metals. After a second, bring the solder into contact with the metals (not the soldering iron itself). It the metals are hot enough, the flux will flow out of the solder and clean the area, then the solder will melt and wet the metals.
5. Remove the solder – pause – then remove the iron without jostling the molten solder left behind.

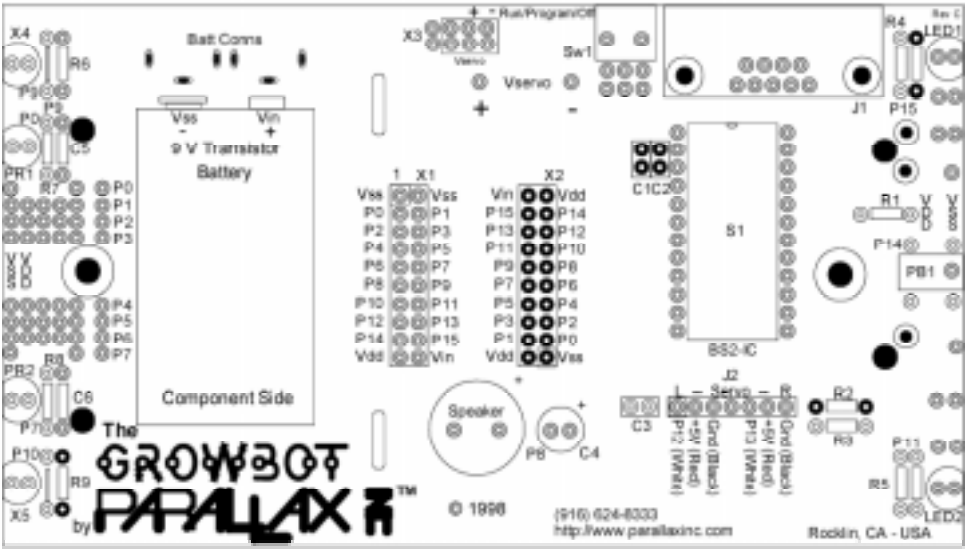
[2] Order part #27951 from Parallax for $34.95, less for educational use.

Chapter 4: GrowBot PCB Assembly

The section shows you how to assemble GrowBot's I/O devices step by step. If you are having difficulty, check the GrowBot section on our web site http://www.parallaxinc.com for updated photographs and pointers. If you are still stuck call our technical support at (916) 624-8333.

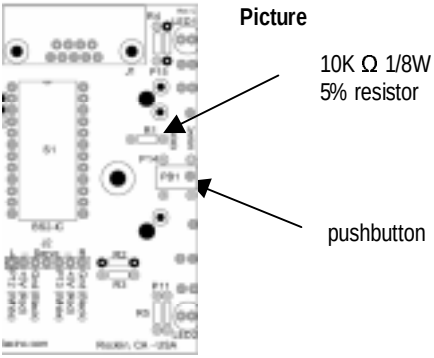
Component locations are labeled on the GrowBot printed circuit board. Soldering is done on the bottom of the GrowBot's printed circuit board (PCB) except where noted. Throughout this documentation references are made to the "front" of the GrowBot. The front is shown below as the left side.

It is very important to trim leads closely to the board so that the battery pack and servos fit tightly against the GrowBot's undercarriage.



Pushbutton Circuit

1. Build the pushbutton circuit by bending the leads one (1) 10k ohm 1/8W 5% resistor (brown-black-orange-gold) and placing it in position R1.
2. Insert the pushbutton into position P14 (PB1). When soldered you can place the red plastic button cover over the pushbutton.
3. Turn the GrowBot PCB over, solder these components and trim the leads.



Battery Monitoring Circuit / Sound Generator

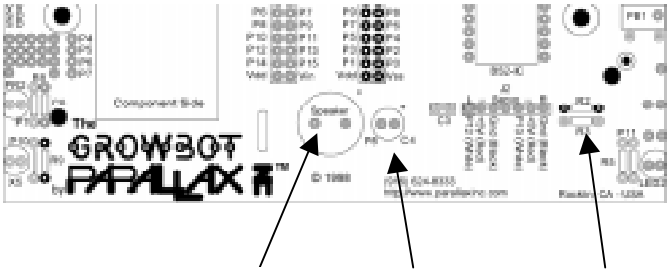
The BS2-IC shares I/O line 8 between the battery monitoring circuit and the sound generator, so it makes sense to build this circuit together.

1. Build the battery monitoring circuit by bending the leads on one (1) 220K ohm resistors and (1) 100K ohm 1/8W 5% resistors and one placing them into positions R2 and R3, respectively.
2. Insert the sound generator and the 10uF 16V aluminum electrolytic capacitor into P8 and C4, respectively. The capacitor's white "-" stripe is away from the PCB's plus sign. The speaker's + lead should be placed accordingly on the board, next to the + sign.
3. Turn the GrowBot PCB over, solder these components and trim the leads.

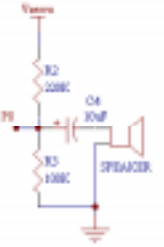
CB Sound Generator Specifications

Rated Voltage (VDC)	Average current consumption (mA)	Sound pressure level (dB)	Resonance frequency
20K	8.5	min 85 typ 89 at 10 cm	4,096 Hz

Picture



Schematic



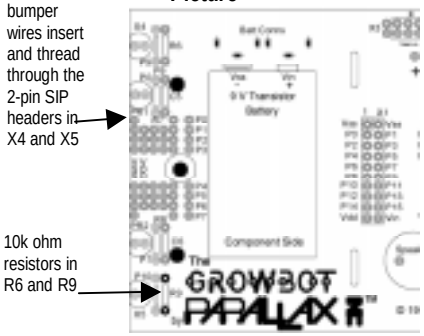
Bumper

The bumper wire is a 0.026 inch diameter stainless steel spring wire covered with insulating shrink tubing. The bumper wire uses the front X4 and X5 I/O locations, with a 2-pin SIP header providing contact closure.

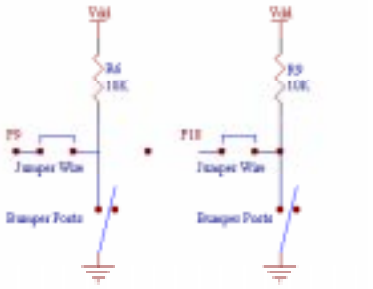
To install the bumper:

1. Insert the two 2-pin SIP headers into the front I/O locations X4 and X5. The two locations are denoted by the outermost round circles.
2. Insert two 10k ohm resistors into positions 6 and R9.
3. Cut off two short leads from a scrap resistor and solder them across P9 and P10.
4. Flip the board over, solder and trim the leads.
5. Insert the ends of the bumper wire into the set of holes next to PR1 and PR2, touching the back portion when the bumper is pushed. Don't solder the end of the bumper wire leads – bend the leads about 45° to keep the bumper in place.

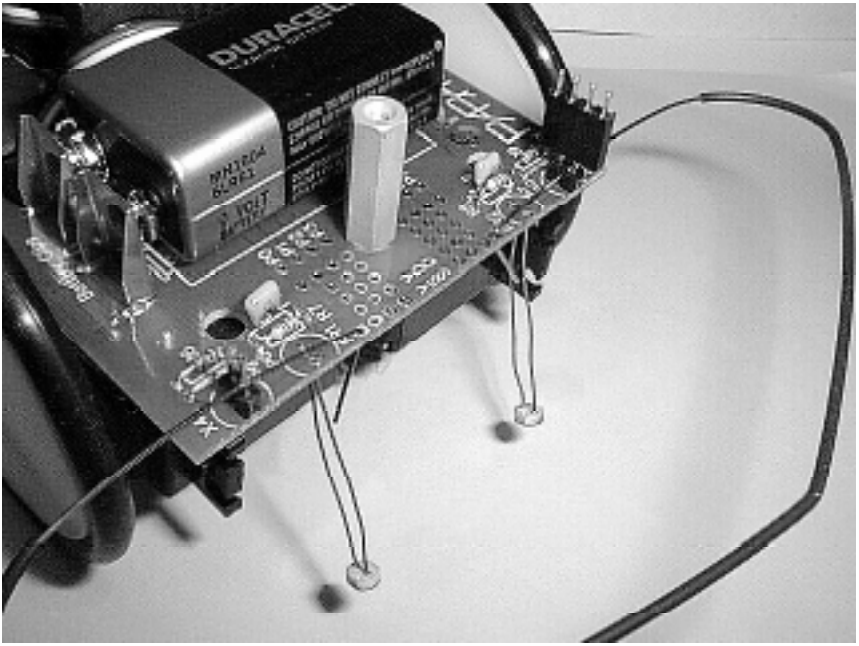
Picture



Schematic



Installed Bumper and Photoresistors



Note the "keeper" plugged onto the X5 position keeps the bumper wires from popping off.

Photoresistors

GrowBot includes two photoconductive cells (photoresistors or LDRs)from EG&G Vactec (part #VT935G group B). The cells provide an inexpensive solution for light level sensing applications. These cells are from Allied Electronics as part number 980-0105. Below are the specifications in case you are interested in getting some more (you can also get these from Parallax):

Photoresistor Specifications

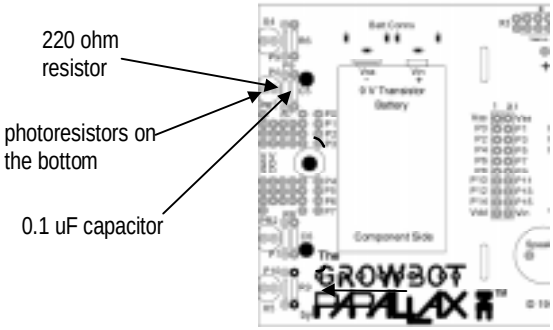
Resistance (Ohms)					Peak Spectral Response nm	V_{MAX}	Response Time @ 1 fc (ms, typ.)	
10 Lux 2850K			Dark					
Min	Typ.	Max.	Min.	Sec.			Rise (1-1/e)	Fall (1/e)
20K	29.0K	38K	1M	10	550	100	35	5

The rover.bs2 program does not use the photoresistors, but edge.bs2 does.

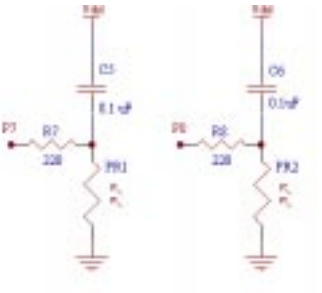
To install the photoresistors:

1. Flip the GrowBot PCB upside down. The photoresistors are installed on the bottom of the PCB, but the capacitor and the resistor are installed on the top.
2. Insert the two photoresistors on the inside I/O ports, PR1 and PR2, respectively. The photoresistors should face forward as shown in the photograph.
3. Turn the GrowBot PCB right side up. Solder and trim the leads.
4. Insert the two 220 ohm resistors into R7 and R8.
5. Insert the two 0.1 µF capacitors into positions C5 and C6.
6. Turn the GrowBot PCB over, solder and trim the leads.

Picture



Schematic



LEDs

The LEDs are installed in two positions: LED1 and LED2. GrowBot uses Liteon LTL-4203 high intensity wide viewing angle LEDs. Any other color or size LED could be substituted provided an appropriately sized current limit resistor is used.

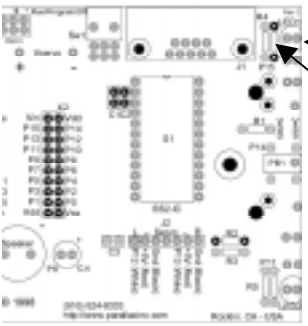
Red LED Specifications

Continuous forward current	Reverse voltage	Forward voltage
40 mA	5 VDC	1.7 VDC at 20 mA

To install the red LEDs:

1. Insert the two 220 ohm resistors into positions R4 and R5.
2. Insert the two red LEDs with the flat side (cathode) facing the edge of the board, fitting within the PCB's footprint outline.
3. Flip the GrowBot board upside down, solder and trim the leads.

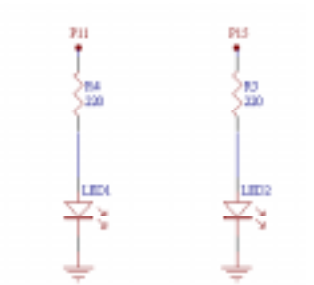
Picture



red LEDs into positions LED1 and LED2

220 ohm resistors in R4 and R5

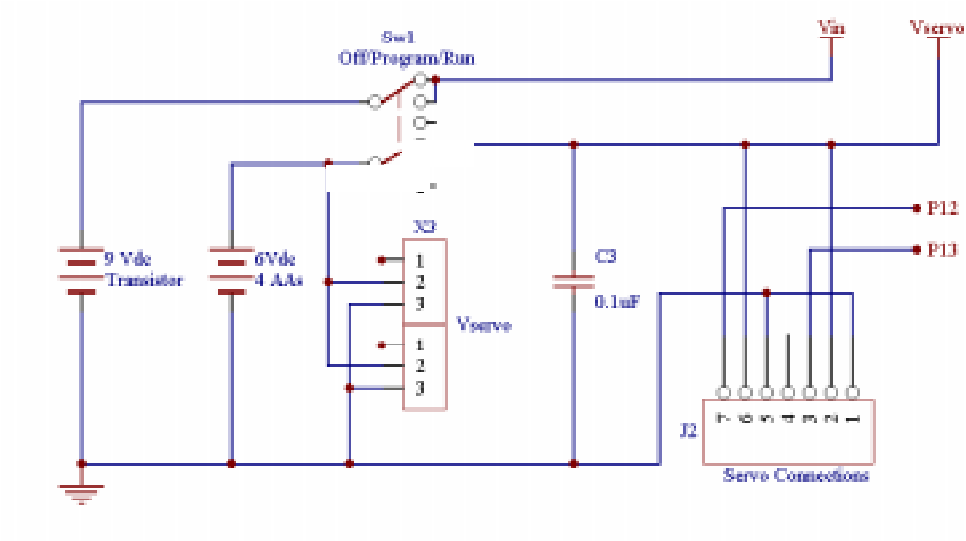
Schematic



Servo and Switch Circuitry

1. Insert two (2) 0.1 µF capacitors into positions C1 and C2. These are for the serial port.
2. Insert one (1) 0.1 µF capacitors into position C3. This capacitor filters the servo power leads.
3. Insert the switch into position SW1.
4. Turn the GrowBot PCB over, solder these components and trim the leads.

Schematic

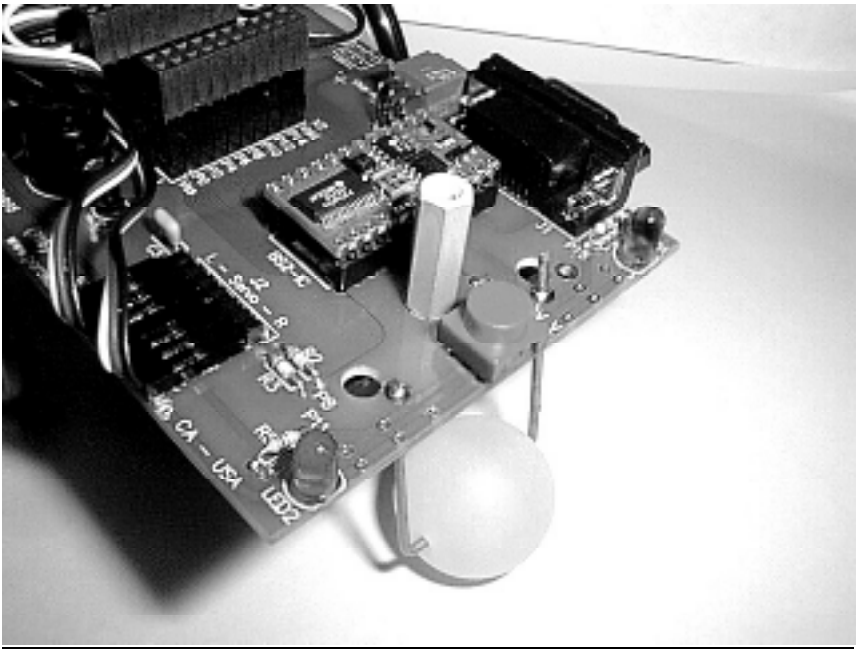


Remaining Hardware

The remaining parts may be soldered onto the PCB once they are inserted in the proper locations.

1. Insert the DB-9 connector into position J1.
2. Insert the 9-volt battery clips.
3. Plug the 24-pin socket into position S1. The semi-circle goes towards the top of the PCB. This will become the BASIC Stamp II's position.
4. Insert the two (2) tall headers into positions X1 and X2. These are for future AppMods in case you want to expamd beyond the basic GrowBot.
5. Insert one (1) 7-pin right angle SIP header into position J2.
6. Turn the board over. Solder these components and trim the leads closely.

Back end of finished GrowBot



Chapter 5: GrowBot PCB Testing

A BS2-IC module was included in this kit. The module has been pre-programmed with a diagnostic program (growtest.bs2) which will show some that some features of the GrowBot's printed circuit board are working at this stage of assembly.

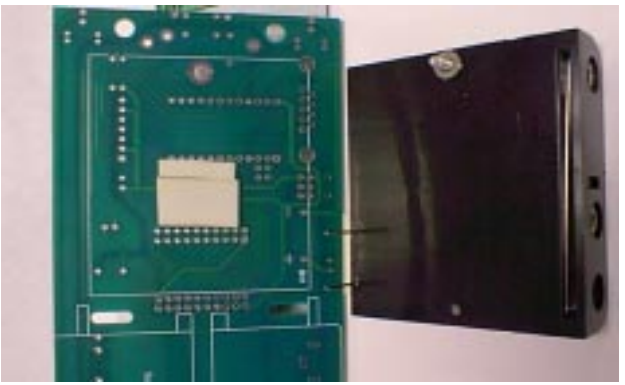
1. Insert the BS2-IC module into socket S1 with Pin 1 by the switch. The biggest chip on the BS2-IC, the PIC 16C57, is closest to the servo connections (if you are using a BS2SX-IC, then the SX28AC/SS is nearest the servo connections).
2. Attach a 9-volt battery to the clips and put the switch in center position. The LEDs should be blinking and the speaker generating a ticking sound. Push the button and the LEDs should alternate as the speaker changes tone frequency for about 10 seconds.

Still doesn't work? Try these GrowBot PCB test fixes. . .
Are the battery clips installed with the correct polarity? The negative connection should be closest to the edge of the board, and the positive connection is closer to the BS2-IC socket.
Is the battery voltage at least 8-volts? Check voltage using a digital multi-meter.
Put the BS2-IC in backwards? Verify that the BS2-IC's Pin 1 is by the switch.
LED polarity must be installed with flat side near the edge of the board.
Speaker and capacitor C4 are installed with correct polarity?
Try reloading the program GROWTEST.BS2.

Servo Battery Holder Installation

1. Attach the double-sided foam tape square to the bottom of the PCB as shown in the photo.
2. Put the screw through the battery holder, and put the nylon washer on the screw.
3. Thread the assembly onto the GrowBot PCB. Screw on the stand-off.
4. Install the other screw and standoff on the front of the board.
5. Solder and trim leads.

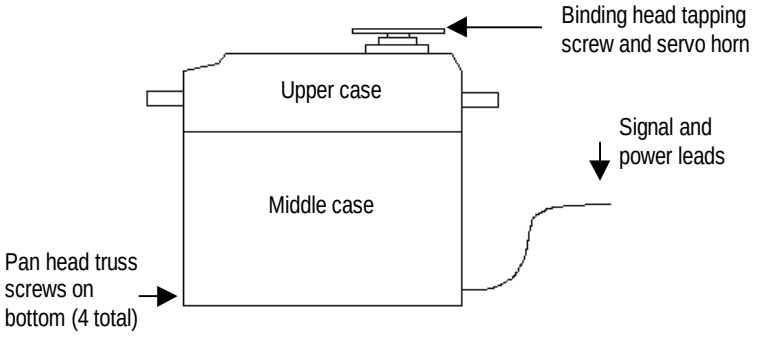
Battery Holder Assembly



Chapter 6: Servo Modification

The GrowBot uses two Futaba S-148 servos for a fully-rotational drivetrain. Futaba servos are standard positioning motors used in R/C applications, with a factory range of motion between 90 and 180 degrees. Servos are only useful for a drivetrain if they can be converted to rotate 360 degrees with reversible rotation. The S-148 is probably the easiest servo to convert, and the procedure is reversible in case you decide to use the servos for something else. Take a look at the S-148. Take a look at the photographs at the end of this chapter.

Futaba S-148 Servo

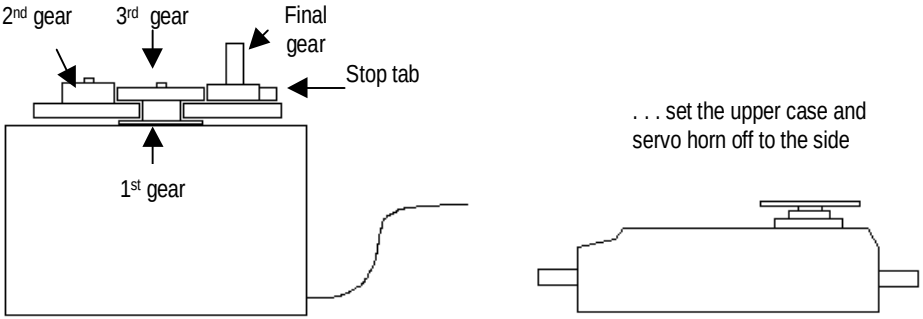

Binding head tapping screw and servo horn
Upper case
Signal and power leads
Middle case
Pan head truss screws on bottom (4 total)

Let's get started with the conversion. These are the first three steps:

1. Loosen the four (4) pan head screws from the bottom.
2. Remove the binding head tapping screw and servo horn from the upper case. Set the upper case aside.
3. Keep the white gears in tact, sitting in position on top of the middle case.

The 1st, 2nd, 3rd, final gear, and metal bearings are now exposed. Its important to keep the parts in tact and remember how it looks because you'll be putting it back together shortly. For consistency, all servo parts are named the same as the Futaba diagram. Internal gears are shown below.

Futaba S-148 Internal Gears

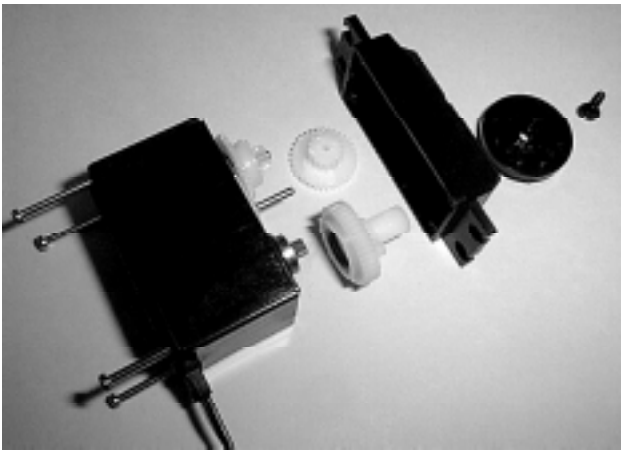


4. Pull of the 3rd gear and set aside.
5. Pull off the final gear and clip the stop tab using cutters. Without this mechanical stop, the servo will be able to rotate 360 degrees.
6. Remove the bearing ring inside of the final gear by prying it out with a small screwdriver.
7. Remove the potentiometer drive plate from the inside of the final gear.
8. Press the bearing ring back inside of the final gear.

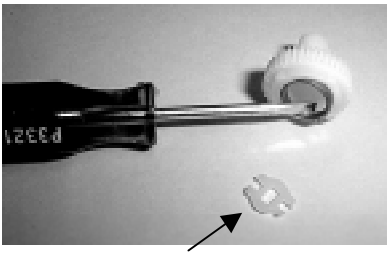
No two servos are exactly alike, which means the GrowBot could wander sideways unless the following mechanical calibrations are made:

9. Plug the servo into the GrowBot PC board connector J2 and turn on the GrowBot (run position all the way to the left). The BS2-IC is programmed to generate a 1.5 ms pulse which corresponds to a center position on a typical servo, or the off condition for the GrowBot. The gears will probably be turning at this point. Turn the brass potentiometer shaft slowly back and forth to find the point where the servo motor stops (or runs slowest) and leave the shaft in that position.
10. Turn off the GrowBot, unplug the servo, reinstall gears and case.

Whew! That was too easy. Let's finish building the GrowBot's PCB.



Clip off the stop tab.



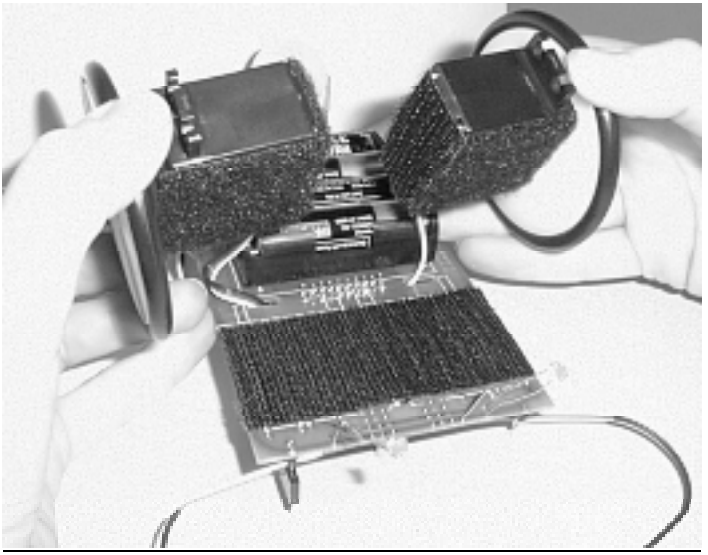
Remove this drive plate.

Chapter 7: Final Assembly

The servos are held on the PCB, and held together by velcro.

1. Cut two fuzzy pieces of velcro 1 ½ inches long and stick them on side of the servos.
2. Cut one fuzzy and one prickly piece of velcro ¾" long. Put one on the bottom of each servo.
3. Cut one prickly piece of velcro 3" long and stick it on the GrowBot's PCB.
4. Attach servos together, and then stick them onto the bottom of the GrowBot's PCB (see picture below).
5. Thread the servo wires through the holes and plug them into their respective positions. Fold the excess wire between the X1 and X2 headers.
6. Push the drive wheels onto the servo shafts and then secure them with the servo screws.
7. Stretch the rubber tires onto the drive wheels.
8. Place the tail wheel wire into the large holes in the PC board and set the Growbot on a flat surface.
9. Level the Growbot by placing something under the battery holder, then solder the tail wheel wire in place.
10. Now you're ready to program your GrowBot.

Servo placement on the bottom of the GrowBot



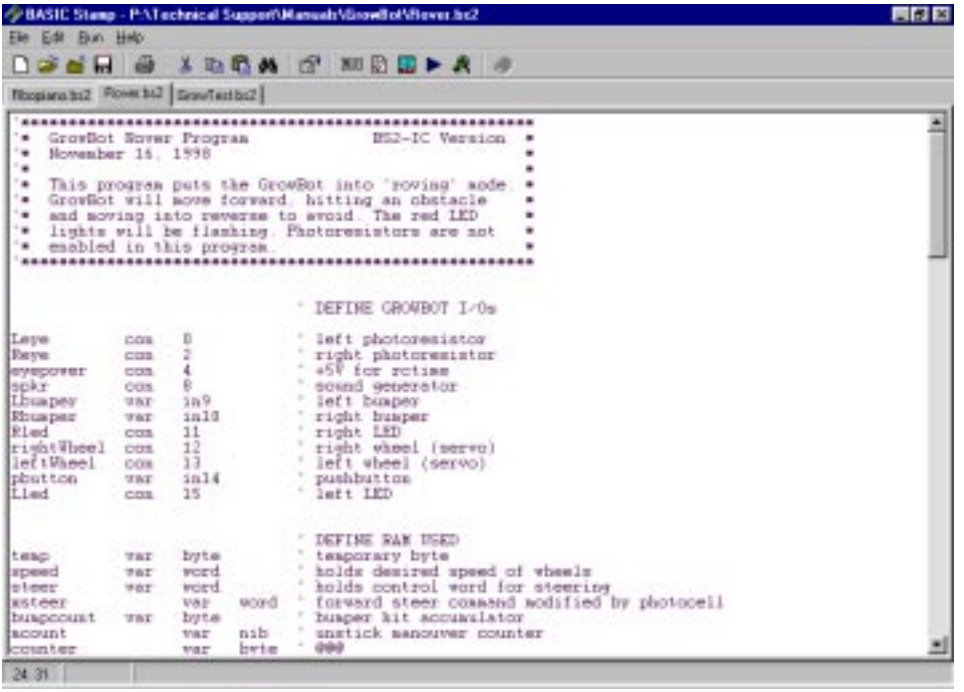
Chapter 8: Program the GrowBot

Run the Editor

BASIC Stamp programs are created and run from a text editor program. The GrowBot disk includes a DOS version (stamp2.exe) and a Win95/98/NT4.0 version (stampw.exe). The use of each editor is similar, but only the Win95/98/NT4.0 interface is discussed in this section. Follow these steps:

1. Using Windows Explorer create a directory on your hard drive named GrowBot.
2. Copy the entire contents of the GrowBot disk into the GrowBot directory.
3. Double-click on the BASIC Stamp II icon (small BS2-IC) to launch the editor program.
4. Under the FILE menu, OPEN rover.bs2. This is a simple roving program.

The screen should look like this:


BASIC Stamp - P:\Technical Support\Manuals\GrowBot\Rover.bs2
File Edit Run Help
'* GrowBot Rover Program BS2-IC Version *
'* November 16, 1998 *
'* This program puts the GrowBot into 'roving' mode. *
'* GrowBot will move forward, hitting an obstacle *
'* and moving into reverse to avoid. The red LED *
'* lights will be flashing. Photoresistors are not *
'* enabled in this program. *
' DEFINE GROWBOT I/Os
Leye con 0 ' left photoresistor
Reye con 2 ' right photoresistor
eyepower con 4 ' +5V for rctime
spkr con 8 ' sound generator
Lbumper var in9 ' left bumper
Rbumper var in10 ' right bumper
Rled con 11 ' right LED
rightWheel con 12 ' right wheel (servo)
leftWheel con 13 ' left wheel (servo)
pbutton var in14 ' pushbutton
Lled con 15 ' left LED
' DEFINE RAM USED
temp var byte ' temporary byte
speed var word ' holds desired speed of wheels
steer var word ' holds control word for steering
xsteer var word ' forward steer command modified by photocell
bumpcount var byte ' bumper hit accumulator
acount var nib ' unstick manouver counter
counter var byte ' ###
24 31

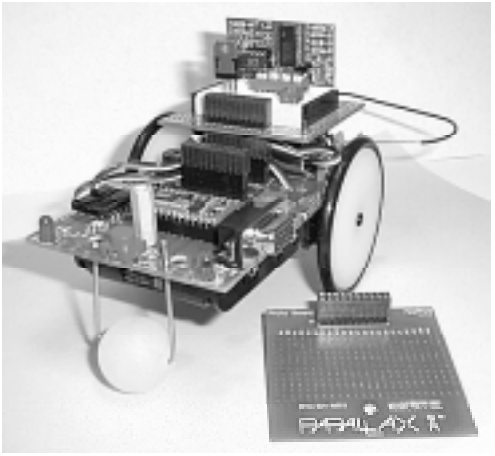
Load the Program into the GrowBot

1. Connect the serial cable to an available serial port on your PC.
2. Connect the other end of the serial cable to your GrowBot's DB-9 connector.
3. Put the GrowBot's power switch in the middle position. This is the "on" position without servos (servos are enabled by switching to the run position).
4. Under the RUN menu, select RUN.

The GrowBot is now programmed to operate in it's roving mode! Push the bumper to see the red LEDs and hear the speaker. Disconnect the serial cable, push the switch to the left position and the GrowBot will be autonomous.

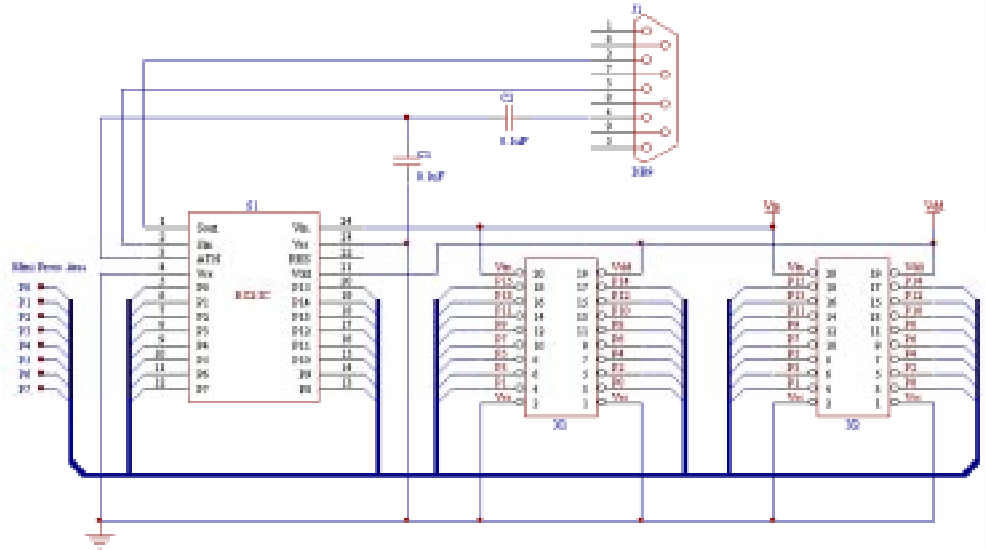
GrowBot programs include a simple rover program using the bumper (rover.bs2); a light / dark sensing edge avoidance program (edge.bs2); and a player piano program adapted from the BASIC Stamp Activity Board ((rbopiano.bs2).

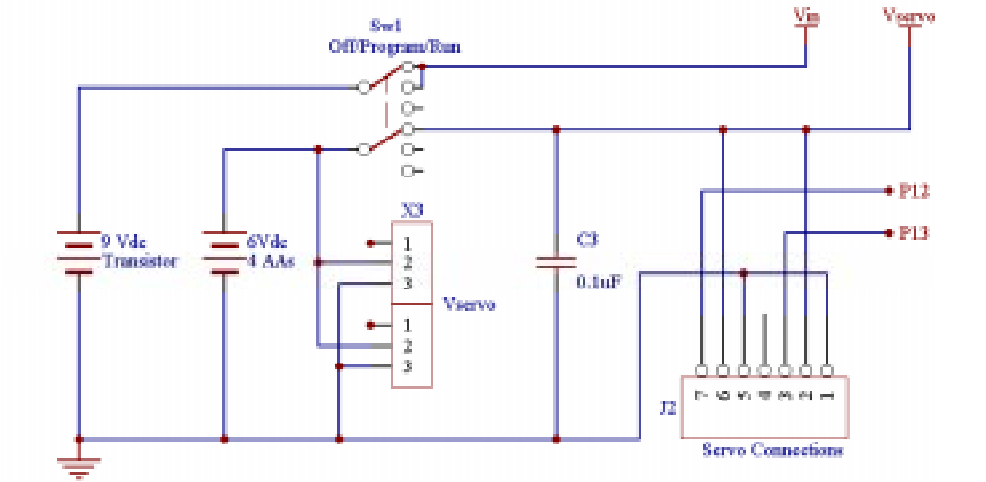
Expand your GrowBot with AppMods

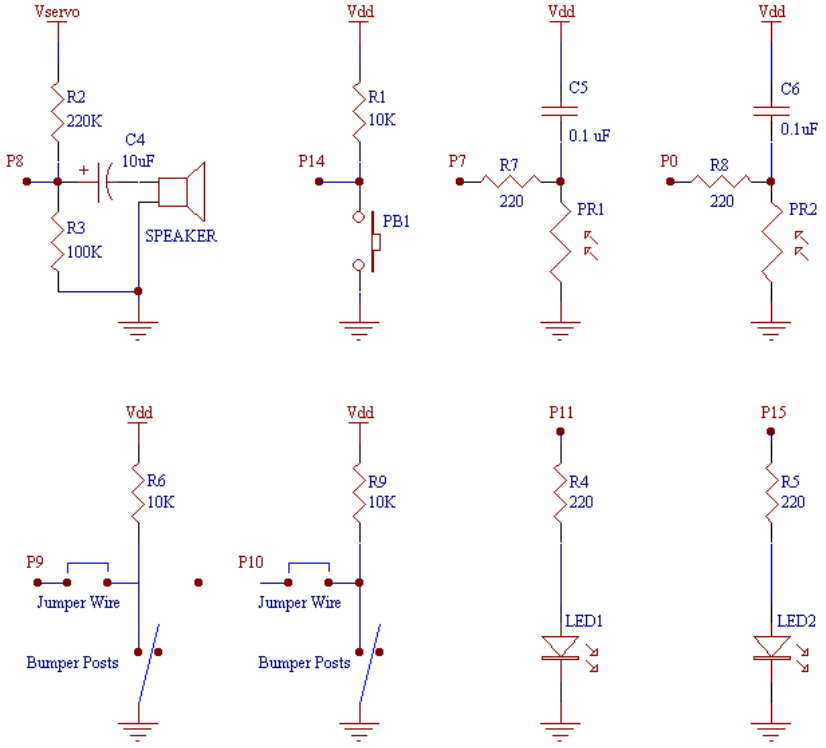


Appendix A: GrowBot Schematic

Each BASIC Stamp II I/O pin is accessible by the AppMod headers, and at solder points on the front of the GrowBot. When using I/Os, watch for conflict with existing on-board devices. These schematics are for the Rev. C GrowBot, December 18, 1998.







GROWBOT BS2 I/O PIN ASSIGNMENTS	
I/O Pin	Use
0	Right photoresistor
7	Left photoresistor
8	Speaker / battery monitor
9	Right bumper
10	Left bumper
11	Left LED
12	Left wheel
13	Right wheel
14	Pushbutton
15	Right LED

Appendix B: GrowBot Programming Tips and PBASIC Quick Reference Guide

The Parallax BASIC Stamp Manual Version 1.9 consists of approximately 450 pages of PBASIC command descriptions, application notes, and schematics. This manual is not included in this GrowBot kit, but it's available for download from http://www.parallaxinc.com in Adobe's PDF format.

BRANCHING

IF...THEN

IF *condition* THEN *addressLabel*

Evaluate condition and, if true, go to the point in the program marked by addressLabel.

- *Condition* is a statement, such as "x=7" that can be evaluated as true of false.
- *AddressLabel* is a label that specifies where to go in the event that the condition is true.

BRANCH

BRANCH *offset, [address0, address1, …address N]*

Go to the address specified by offset (if in range).

- *Offset* is a variable / constant that specifies which of the listed address to go to (0-N).
- *Addresses* are labels that specify where to go.

GOTO

GOTO *addressLabel*

Go to the point in the program specified by addressLabel.

- *AddressLabel* is a label that specifies where to go.

GOSUB

GOSUB *addressLabel*

Store the address of the next instruction after GOSUB, then go to the point in the program specified by addressLabel.

- *AddressLabel* is a label that specifies where to go.

RETURN

Return from subroutine – sends the program back to the address (instruction) immediately following the most recent GOSUB.

LOOPING

FOR...NEXT

FOR *variable = start to end {stepVal} …NEXT*

Create a repeating loop that executes the program lines between FOR and NEXT, incrementing or decrementing *variable* according to *stepVal* until the value of the variable pases the *end* value.

- *Variable* is a bit, nib, byte, or word variable used as a counter.
- *Start* is a variable or constant that specifies the initial value of the variable.
- *End* is a variable or constant that specifies the end value of the variable. When the value of the variable passes *end*, the FOR . . . NEXT loop stops executing and the program goes on to the instruction after NEXT.
- *StepVal* is an optional variable or constant by which the variable increases or decreases with eachteip through the FOR / NEXT loop. If *start* is larger than *end*, PBASIC2 understands *stepVal* to be negative, even though no minus sign is used.

NUMERICS

LOOKUP

LOOKUP index, [value0, value1,... valueN], variable

Look up the value specified by the index and store it in a variable. If the index exceeds the highest index value of the items in the list, variable is unaffected. A maximum of 256 values can be included in the list.

- *index* is a constant, expression or a bit, nibble, byte or word variable.
- *value0*, *value1,* etc. are constants, expressions or bit, nibble, byte or word variables.
- *variable* is a bit, nibble, byte or word variable.

LOOKDOWN

LOOKDOWN *value, {??,} [value0, value1,... valueN], variable*

- *value* is a constant, expression or a bit, nibble, byte or word variable.
- *??* is =, <>, >, <, <=, =>. (= is the default).
- *value0, value1*, etc. are constants, expressions or bit, nibble, byte or word variables.
- *variable* is a bit, nibble, byte or word variable.

RANDOM

RANDOM *variable*

Generate a pseudo-random number.

- VARIABLE is a byte or word variable in the range 0..65535.

DIGITAL I/O

INPUT

INPUT *pin*

Make the specified pin an input (write a 0 to the corresponding bit of DIRS).

- *pin* is a constant, expression or a bit, nibble, byte or word variable in the range 0..15.

OUTPUT

OUTPUT *pin*

Make the specified pin an output (write a 1 to the corresponding bit of DIRS).

- *pin* is a constant, expression or a bit, nibble, byte or word variable in the range 0..15.

REVERSE

REVERSE *pin*

If pin is an output, make it an input. If pin is an input, make it an output.

- *pin* is a constant, expression or a bit, nibble, byte or word variable in the range 0..15.

LOW

LOW *pin*

Make pin output low (write 1 to the corresponding bit of DIRS and 0 to the corresponding bit of OUTS).

- *pin* is a constant, expression or a bit, nibble, byte or word variable in the range 0..15.

HIGH

HIGH *pin*

Make the specified pin output high (write 1s to the corresponding bits of both DIRS and OUTS).

- *pin* is a constant, expression or a bit, nibble, byte or word variable in the range 0..15.

TOGGLE

TOGGLE *pin*

Invert the state of a pin.

- *pin* is a constant, expression or a bit, nibble, byte or word variable in the range 0..15.

PULSIN

PULSIN *pin, state, variable*

Measure an input pulse (resolution of 2 µs).

- *pin* is a constant, expression or a bit, nibble, byte or word variable in the range 0..15.
- *state* is a constant, expression or a bit, nibble, byte or word variable in the range 0..1.
- *variable* is a bit, nibble, byte or word variable.
- Measurements are in 2uS intervals and the instruction will time out in 0.13107 seconds.

PULSOUT

PULSOUT *pin, period*

Output a timed pulse by inverting a pin for some time (resolution of 2 µs).

- *pin* is a constant, expression or a bit, nibble, byte or word variable in the range 0..15.
- *period* is a constant, expression or a bit, nibble, byte or word
- *variable* in the range 0..65535 representing the pulse width in 2uS units.

BUTTON

BUTTON *pin, downstate, delay, rate, workspace, targetstate, label*

Debounce button, perform auto-repeat, and branch to address if button is in target state.

- *pin* is a constant, expression or a bit, nibble, byte or word variable in the range 0..15.
- *downstate* is a constant, expression or a bit, nibble, byte or word variable in the range 0..1.
- *delay* is a constant, expression or a bit, nibble, byte or word variable in the range 0..255.
- *rate* is a constant, expression or a bit, nibble, byte or word variable in the range 0..255.
- *workspace* is a byte or word variable.
- *targetstate* is a constant, expression or a bit, nibble, byte or word variable in the range 0..1.
- *label* is a valid label to jump to in the event of a button press.

SHIFTIN

SHIFTIN *dpin, cpin, mode, [result{\bits} { ,result{\bits}... }]*

Shift bits in from parallel-to-serial shift register.

- *dpin* is a constant, expression or a bit, nibble, byte or word variable in the range 0..15 specifying the data pin.
- *cpin* is a constant, expression or a bit, nibble, byte or word variable in the range 0..15 specifying the clock pin.
- *mode* is a constant, symbol, expression or a bit, nibble, byte or word variable in the range 0..4 specifying the bit order and clock mode. 0 or MSBPRE = msb first, pre-clock, 1 or LSBPRE = lsb first, pre-clock, 2 or MSBPOST = msb first, post-clock, 3 or LSBPOST = lsb first, post-clock.
- *result* is a bit, nibble, byte or word variable where the received data is stored.
- *bits* is a constant, expression or a bit, nibble, byte or word variable in the range 1..16 specifying the number of bits to receive in *result.* The default is 8.

SHIFTOUT

SHIFTOUT *dpin, cpin, mode, [data{\bits} {, data{\bits}... }]*

Shift bits out to serial-to-parallel shift register.

- *dpin* is a constant, expression or a bit, nibble, byte or word variable in the range 0..15 specifying the data pin.
- *cpin* is a constant, expression or a bit, nibble, byte or word variable in the range 0..15 specifying the clock pin.
- *mode* is a constant, symbol, expression or a bit, nibble, byte or word variable in the range 0..1 specifying the bit order. 0 or LSBFIRST = lsb first, 1 or MSBFIRST = msb first.
- *data* is a constant, expression or a bit, nibble, byte or word variable containing the data to send out.
- *bits* is a constant, expression or a bit, nibble, byte or word variable in the range 1..16 specifying the number of bits of *data* to send. The default is 8.

COUNT

COUNT *pin, period, result*

Count cycles on a pin for a given amount of time (0 - 125 kHz, assuming a 50/50 duty cycle).

- *pin* is a constant, expression or a bit, nibble, byte or word variable in the range 0..15.
- *period* is a constant, expression or a bit, nibble, byte or word variable in the range 0..65535.
- *result* is a bit, nibble, byte or word variable.

XOUT

XOUT *mpin, zpin, [house\keyorcommand{\cycles} {, house\keyorcommand{\cycles}... }]*

Generate X-10 powerline control codes. For use with TW523 or TW513 powerline interface module.

- *mpin* is a constant, expression or a bit, nibble, byte or word variable in the range 0..15 specifying the modulation pin.
- *zpin* is a constant, expression or a bit, nibble, byte or word variable in the range 0..15 specifying the zero-crossing pin.
- *house* is a constant, expression or a bit, nibble, byte or word variable in the range 0..15 specifying the house code A..P respectively.
- *keycommand* is a constant, expression or a bit, nibble, byte or word variable in the range 0..15 specifying keys 1..16 respectively or is one of the commands in the following table:
- See the Parallax manual for a list of X-10 commands for lights on, off, dim and bright.
- *cycles* is a constant, expression or a bit, nibble, byte or word variable in the range 2..65535 specifying the number of cycles to send. (Default is 2).

SERIAL I/O

SERIN

SERIN *rpin{\fpin}, baudmode, {plabel,} {timeout, tlabel,} [inputdata]*

Serial input with optional qualifiers, time-out, and flow control. If qualifiers are given, then the instruction will wait until they are received before filling variables or continuing to the next instruction. If a time-out value is given, then the instruction will abort after receiving nothing for a given amount of time. Baud rates of 300 - 50,000 are possible (0 - 19,200 with flow control). Data received must be N81 (no parity, 8 data bits, 1 stop bit) or E71 (even parity, 7 data bits, 1 stop bit).

- *rpin* is a constant, expression or a bit, nibble, byte or word variable in the range 0..16.
- *fpin* is a constant, expression or a bit, nibble, byte or word variable in the range 0..15.
- *baudmode* is a constant, expression or a bit, nibble, byte or word variable in the range 0..65535.
- *plabel* is a label to jump to in case of a parity error.
- *timeout* is a constant, expression or a bit, nibble, byte or word variable in the range 0..65535 representing the number of milliseconds to wait for an incoming message.
- *tlabel* is a label to jump to in case of a timeout.
- *inputdata* is a set of constants, expressions and variable names separated by commas and optionally proceeded by the formatters available in the DEBUG command, except the ASC and REP formatters. Additionally, the following formatters are available:
    1. STR bytearray\L{\E} input a string into bytearray of length L with optional end-character of E. (0's will fill remaining bytes).
    2. SKIP L input and ignore L bytes.
    3. WAITSTR bytearray{\L} Wait for bytearray string (ofL length, or terminated by 0 if parameter is not specified and is 6 bytes maximum).
    4. WAIT (value {,value...}) Wait for up to a six-byte se-quence.

SEROUT

SEROUT *tpin{\fpin}, baudmode, {pace,} {timeout, tlabel,} [outputdata]*

Send data serially with optional byte pacing and flow control. If a pace value is given, then the instruction will insert a specified delay between each byte sent (pacing is not available with flow control). Baud rates of 300 - 50,000 are possible (0 - 19,200 with flow control). Data is sent as N81 (no parity, 8 data bits, 1 stop bit) or E71 (even parity, 7 data bits, 1 stop bit).

- *tpin* is a constant, expression or a bit, nibble, byte or word variable in the range 0..16.
- *fpin* is a constant, expression or a bit, nibble, byte or word variable in the range 0..15.
- *baudmode* is a constant, expression or a bit, nibble, byte or word variable in the range 0..60657.
- *pace* is a constant, expression or a bit, nibble, byte or word variable in the range 0..65535 specifying a time (in milliseconds) to delay between transmitted bytes. This value can only be specified if the *fpin* is not specified.
- *timeout* is a constant, expression or a bit, nibble, byte or word variable in the range 0..65535 representing the number of milliseconds to wait for the signal to transmit the message. This value can only be specified if the *fpin* is specified.
- *tlabel* is a label to jump to in case of a timeout. This can only be specified if the *fpin* is specified.
- *outputdata* is a set of constants, expressions and variable names separated by commas and optionally proceeded by the formatters available in the DEBUG command.

ANALOG I/O

PWM

PWM *pin, duty, cycles*

Output PWM, then return pin to input. This can be used to output analog voltages (0-5V) using a capacitor *and resistor.*

- *pin* is a constant, expression or a bit, nibble, byte or word variable in the range 0..15.
- *duty* is a constant, expression or a bit, nibble, byte or word variable in the range 0..255.
- *cycles* is a constant, expression or a bit, nibble, byte or word variable in the range 0..255 representing the number of 1ms cycles to output.

RCTIME

RCTIME *pin, state, variable*

Measure an RC charge/discharge time. Can be used to measure potentiometers.

- *pin* is a constant, expression or a bit, nibble, byte or word variable in the range 0..15.
- *state* is a constant, expression or a bit, nibble, byte or word variable in the range 0..1.
- *variable* is a bit, nibble, byte or word variable.

SOUND

FREQOUT

FREQOUT *pin, milliseconds, freq1 {,freq2}*

Generate one or two sinewaves of specified frequencies (each from 0 - 32767 hz.).

- *pin* is a constant, expression or a bit, nibble, byte or word variable in the range of 0..15.
- *milliseconds* is a constant, expression or a bit, nibble, byte or word variable.
- *freq1* and *freq2* are constant, expression or bit, nibble, byte or word variables in the range 0..32767 representing the correspond-ing frequencies.

DTMFOUT

DTMFOUT *pin, {ontime, offtime,}[key{,key...}]*

Generate DTMF telephone tones.

- *pin* is a constant, expression or a bit, nibble, byte or word variable in the range 0..15.
- *ontime* and *offtime* are constants, expressions or bit, nibble, byte or word variables in the range 0..65535.
- *key* is a constant, expression or a bit, nibble, byte or word variable in the range 0..15.

EEPROM ACCESS

DATA

DATA {pointer} DATA {@location,} {WORD} {data}{(size)} {, { WORD} {data}{(size)}...}

Store data in EEPROM before downloading PBASIC program.

- *pointer* is an optional undefined constant name or a bit, nibble, byte or word variable which is assigned the value of the first memory location in which data is written.
- *location* is an optional constant, expression or a bit, nibble, byte or word variable which designates the first memory location in which data is to be written.
- *word* is an optional switch which causes DATA to be stored as two separate bytes in memory.
- *data* is an optional constant or expression to be written to memory.
- *size* is an optional constant or expression which designates the number of bytes of defined or undefined data to write/reserve in memory. If DATA is not specified then undefined data space is reserved and if DATA is specified then SIZE bytes of data equal to DATA are written to memory.

READ

READ *location, variable*

Read EEPROM byte into variable.

- *location* is a constant, expression or a bit, nibble, byte or word variable in the range 0..2047.
- *variable* is a bit, nibble, byte or word variable.

WRITE

WRITE *location, data*

Write byte into EEPROM.

- *location* is a constant, expression or a bit, nibble, byte or word variable in the range 0..2047.
- *data* is a constant, expression or a bit, nibble, byte or word variable.

TIME

PAUSE

PAUSE *milliseconds*

Pause execution for 0–65535 milliseconds.

- *milliseconds* is a constant, expression or a bit, nibble, byte or word variable in the range 0..65535.

POWER CONTROL

NAP

NAP *period*

Nap for a short period. Power consumption is reduced.

- *period* is a constant, expression or a bit, nibble, byte or word variable in the range 0..7 representing 18ms intervals.
- Current is reduced to 50uA (assuming no loads).

SLEEP

SLEEP *seconds*

Sleep for 1-65535 seconds. Power consumption is reduced to approximately 50 µA.

- *seconds* is a constant, expression or a bit, nibble, byte or word variable in the range 0..65535 specifying the number of seconds to sleep.

END

END

Sleep until the power cycles or the PC connects. Power consumption is reduced to approximately 50 µA.

- 50uA reduced current (no loads).

PROGRAM DEBUGGING

DEBUG

DEBUG *outputdata{,outputdata...}*

Send variables to PC for viewing.

- *outputdata* is a text string, constant or a bit, nibble, byte or word variable. If no formatters are specified DEBUG defaults to ascii character display without spaces or carriage returns following the value.